Attorney Docket: 02998.P011 *PATENT*

UNITED STATES PATENT APPLICATION

FOR

LOGIC SYSTEM WITH CONFIGURABLE INTERFACE

INVENTORS

DREW WINGARD

MICHAEL J. MEYER

GEERT ROSSEEL

LISA A. ROBINSON

JAY S. TOMLINSON

PREPARED BY:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1026
(408) 720-8598

"Express Mail" mailing label number EL234218166US

Date of Deposit: Aug. 8, 2000

I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, Washington, D.C. 20231

Cheri Clinkenbeard

(Typed or printed name of person mailing paper or fee)

C Clinkenbeard 8/8/00

(Signature of person mailing paper or fee) *Date*

LOGIC SYSTEM WITH CONFIGURABLE INTERFACE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to a configurable interface enabling straightforward re-use of a core with different interfaces.

2. Art Background

The latest advances in semiconductor technology and design methodology have enabled the emerging market for System-On-a-Chip (SoC) designs. Full systems, consisting of more than several million logic gates, can now be implemented in a single chip. One of the main design challenges in these SoC designs is the logical and physical interconnect that allows communication between the subsystem cores that compose the design. These cores typically fall into different categories: computing cores such as a CPU (central processing unit), DSP (digital signal processor) or floating point co-processor; peripheral interface cores such as PCI (personal computer interface) or USB (universal serial bus); memory blocks such as SRAM (static random access memory) and on-chip DRAM (dynamic random access memory); and application specific blocks such as video cores (MPEG-motion pictures experts group) or communication cores.

Since many of the SoC designs are targeted towards communications and consumer applications, time-to-market is a critical factor in the decision process on the level of integration to be used in a particular product. Once a core has been proven in one design, it becomes very attractive to re-use the core in later designs. While choosing a proven design may eliminate the time that would otherwise be required to design a new core, design re-use offers the promise of many other benefits. First, a model may be written for the proven core that can provide accurate results when analyzing the requirements and performance of a

new system design; the model for a new, unproven core is likely to be neither as accurate as the proven core, nor built in time to influence the design. Second, proven cores can serve as building blocks that simplify the overall design process by allowing the system designer to focus at a higher level of abstraction, while providing improved predictability in the resulting system implementation. Third, re-use of hardware cores protects the investment in software to control those cores, and allows the system software implementation to proceed as soon as the hardware building blocks have been chosen. Finally, core re-use protects the investment in verification and testing. Since the desired systems are highly integrated, the required cores end up deeply embedded within an integrated circuit. In deeply-embedded designs, verifying the design functionality becomes very challenging and testing an individual system to prove that it is correctly built can lead to expensive delays or costly system rework. Thus, maintaining the integrity of core verification and testing is likely the single biggest gain from design re-use.

However, historically, re-using cores has not been efficient. One of the challenges in the re-use of these cores is that, dependent on the system requirements of the SoC that the core is used in, different performance levels and features are required from the core. Moreover, many applications of SoC are targeted towards consumer applications where providing the most cost-effective solution is very important.

The interface of the core has been a source of inefficiency in core re-use. **Figure 1a** shows a representation of a core with a particular interface. Inefficiencies arise when this core must be re-used in a different system that requires a different interface of the core, or demands different requirements of the core.

One option is to re-design the interface of the core. This solution may offer area and performance benefits, but is expensive in term of time, effort and design risk. The effort involved is not only in terms of design time but there is also

significant additional work in verification and validation. This solution is represented in **Figure 1b**, which shows the core with the optimized interface.

Rather than changing the core, some designers opt to leave the core as is, and adapt the system level interconnect to the existing core interface. While it preserves the integrity of the original core, it leads to many other inefficiencies. With respect to performance, the logic that integrates the core into the system can add latency into the system, which can adversely affect the system performance. With respect to cost, the additional logic can add a significant number of gates to the design and hence increase the chip area and hence the cost. This solution is represented in **Figure 1c**.

SUMMARY

The system and method of the present invention provides a core or subsystem with a configurable interface that enables straightforward re-use of the core. In one embodiment, code representative of the core and configurable interface parameters are combined with input consisting of the defined configurable interface parameters to generate a core having an interface configured in accordance with the defined interface parameters.

BRIEF DESCRIPTION OF THE DRAWINGS

The objects, features and advantages of the present invention will be apparent from the following detailed description in which:

Figure 1a, 1b, and 1c illustrate prior art techniques relating to core interfacing.

Figure 2a and 2b illustrate one embodiment of a logic interface.

Figure 3a, 3b, and 3c illustrate embodiments of configurable interface parameters.

Figure 4 illustrates one embodiment of an interface generated in accordance with the teachings of the present invention.

Figure 5 is an example of a configuration file.

Figure 6 illustrates one embodiment of a process for generating a core using a previously configured interface.

Figure 7 is an example of a graphical user interface utilized in one embodiment to select a configuration of the interface.

DETAILED DESCRIPTION

In the system and the method of the present invention, the above challenges of creating optimal cores for a particular system are resolved by implementing a core with a highly configurable interface, such that the core together with its interface can be optimally configured for the particular system that the core is used in. In the following description, for purposes of explanation, numerous details are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that these specific details are not required in order to practice the present invention. In other instances, well-known electrical structures and circuits are shown in block diagram form in order not to obscure the present invention unnecessarily.

The system and the method of the present invention will be explained by example, initially referring to **Figure 2a**. **Figure 2a** shows exemplary signals of a simple logical interface. The signals include a request phase that includes command, address, data and command accept, and a response phase that includes response and data. **Figure 2b** illustrates an interface providing the connection between two cores. For purposes of discussion herein cores are defined as logic or circuitry that performs a function or functions that receives input and/or generates output at least in part through a configurable interface.

In one embodiment, the function to be performed can be specified by the MCmd lines of **Figure 2b**. One embodiment of the MCmd encoding specifies read, write and exclusive read functions, as shown in **Figure 3a**. Such functions may represent typical operations performed in computer systems. For instance the exclusive read may indicate that a read to a specific address must be followed by a write to the same address before any other core in the system can read that location.

In one embodiment, the command encoding can be as given in **Figure 3a**. The encoded functions are functions that are typically used in computer systems.

Exclusive Read indicates that a read to a specific address must be followed by a write to the same address before any other core in the system can write that location.

The above interface illustrated by **Figure 2a and 2b and 3a** is a simple interface and can be used when the core only needs to support low-performance requirements from the system. One can now extend the interface to incorporate additional functional and performance related features in a configurable manner such that the core support many different combinations on interface options. In alternate embodiments, fewer or additional types of configurability may be implemented. In one embodiment herein, there are three different types of configurability.

In an exemplary type of configuration, one can configure the width of a particular field. As an example, the width of the address field can be configured to be a value between 1 and 32 lines. This allows the core to be used in systems that require different sizes of address space. This type of configurability is referred to herein as "parametrization".

In an exemplary second type of configuration, one can select the availability of certain interface functions. In the command-encoding example of **Figure 3a**, one can make the function "Exclusive Read" a configurable option, such that this function can be enabled when the core is used in a system that requires this. This type of configurability is referred to herein as "function enabling".

In an exemplary third type of configuration, a signal can be configured to be present or not. This type of configurability is referred to herein as "signal-enabling". **Figure 2b** represents a very simple logical interface, which can be used for cores with low performance requirements. A higher level of interface functionality and complexity may include the addition of a burst field (Mburst),

indicating that the addresses of subsequent commands are logically related. One embodiment of the burst encoding of the Mburst signal is shown in **Figure 3b**.

As indicated herein, an incrementing burst indicates a command sequence where the address increments by the number of bytes in the data word with every new command being issued in the burst. A non-incrementing burst is a burst sequence where the address remains unchanged between the commands in the burst. A core with burst configured in potentially allows much higher read and write throughput for transferring a large block of data. Again, this option can be optionally configured into the core interface if the core is used in a system that can take advantage of this feature.

The greater the configurability of the interface, the wider the usability of the core. One embodiment of an extended and highly configurable core interface is shown in **Figure 4**.

Some of the extensions illustrated by **Figure 4** include the ByteEnable Field (MByteEn) indicates which bytes in the MData Field are to be read or written; MError and SError that represent error signals; and MFlag and SFlag fields that are used for transferring out-of-band information between the core and the system. A traditional and well-known example of this kind of information is synchronization, where for example, the system is waiting for an out-of-band signal from the core before the system transmits any further requests.

One embodiment of a method of generating the optimal core is illustrated in **Figure 6**. At step 610, the core source code with at least one configurable interface parameter is provided. In one embodiment, for each interface configuration option, a parameter is defined, together with a range of allowable values. For example, for configuring the width the MData field of **Figure 4**, the parameter name MData_WIDTH is defined and the allowable values are 8,16,32 and 64. For signal-enabling the MBurst field, the parameter MBurst_ENABLE

can be defined with the allowable values of 0 and 1, where the value 0 indicates that it is not present and the value of indicates that it is.

In one embodiment, the core is implemented as configurable source code that makes use of these parameters or derived versions of these parameters. This source code can be in a variety of forms of e.g. commercially available hardware description languages (Verilog, VHDL) or software languages (C, perl, ...), or any combination of these or any other language.

At step 620, the configuration settings are provided. In one embodiment, the configuration settings are defined in a machine-readable form. In one embodiment, the configuration settings for a particular core are defined in a file. **Figure 5** shows an example of a configuration file describing a particular configuration of the interface described in **Figure 4**.

At step 625, the source code and configurations settings are combined, e.g., compiled, to generate the core with the configured interface. In one embodiment, a software program, referred to herein as the core compiler, process, step 625, the configurable source code representation of the core is combined with the data of the configuration to generate a core with the desired interface, step 630.

In one embodiment, the configuration settings can be entered manually by the user; alternatively the settings can be entered through a Graphical User Interface. **Figure 7** illustrates one embodiment of an example Graphical User Interface for configuring a set of options on a core with a configurable interface similar to the one described in **Figure 4**. These values, which are described by the text in the GUI window, are used by software, together with the configurable source code, to derive the core with the desired interface.

The invention has been described in conjunction with the preferred embodiment. It is evident that numerous alternatives, modifications, variations and uses will be apparent to those skilled in the art in light of the foregoing description.